Code pour les capteurs infrarouges par Kambale kambere leonce Engineer.

```
// Feu Tricolore Intelligent 4 Voies - Capteurs IR
// Comptage + Vitesse + Priorité Cortège + Piétons

// --- CONSTANTES TEMPS ---
const unsigned long T_VERT_MIN = 60000; // 60s
const unsigned long T_JAUNE_MIN = 20000; // 20s
const unsigned long T_ROUGE_MIN = 60000; // 60s
const unsigned long T_ADD_PAR_VOITURE = 3000; // +3s par voiture

// --- CONSTANTES DETECTION ---
const float DISTANCE_ENTRE_CAPTEURS = 2.0; // 2 mètres entre les 2 IR
const float VITESSE_CORTEGE_BASSE = 10.0; // km/h deuil/mariage
const float VITESSE_CORTEGE_HAUTE = 60.0; // km/h autorités

// --- STRUCTURE VOIE ---
struct Voie {
  // LEDs Feu
  int pinV, pinJ, pinR;
  // Capteurs IR: [0]=entrée, [1]=sortie pour vitesse
  int pinIR1, pinIR2;

  int compteurVoitures;
```

```
  bool priorite;
  unsigned long tempsVert;

  // Variables internes vitesse
  unsigned long t1; // temps coupure capteur 1
  bool voitureEnCours;
};

Voie voies[4];
int voieActive = 0; // 0=A, 1=B, 2=C, 3=D
unsigned long chronoDepartVert;

// Pins Piétons
const int btnPieton1 = 2;
const int btnPieton2 = 3;
const int ledPieton1 = 46;
const int ledPieton2 = 47;

void setup() {
  Serial.begin(9600);

  // Init Voie A Route 1
  voies[0] = {22,23,24, 30,31, 0,false,T_VERT_MIN, 0,false};
  // Init Voie B Route 1
  voies[1] = {25,26,27, 32,33, 0,false,T_VERT_MIN, 0,false};
```

```
  // Init Voie C Route 2
  voies[2] = {28,29,30, 34,35, 0,false,T_VERT_MIN, 0,false};
  // Init Voie D Route 2
  voies[3] = {31,32,33, 36,37, 0,false,T_VERT_MIN, 0,false};

  // Init Pins
  for(int i=0; i<4; i++){
    pinMode(voies[i].pinV, OUTPUT);
    pinMode(voies[i].pinJ, OUTPUT);
    pinMode(voies[i].pinR, OUTPUT);
    pinMode(voies[i].pinIR1, INPUT_PULLUP); // IR donne LOW quand obstacle
    pinMode(voies[i].pinIR2, INPUT_PULLUP);
  }

  pinMode(btnPieton1, INPUT_PULLUP);
  pinMode(btnPieton2, INPUT_PULLUP);
  pinMode(ledPieton1, OUTPUT);
  pinMode(ledPieton2, OUTPUT);

  toutRouge();
  delay(2000);
  passerAuVert(0); // Démarre sur voie A
}

void loop() {
```

```
  // 1. LIRE TOUS LES CAPTEURS
  for(int i=0; i<4; i++){
    lireCapteursIR(i);
  }

  // 2. GESTION PIETONS
  if(digitalRead(btnPieton1)==LOW || digitalRead(btnPieton2)==LOW){
    gererPietons();
  }

  // 3. GESTION DU CYCLE
  gererCycle();
}

// --- LECTURE CAPTEURS IR ---
void lireCapteursIR(int i){
  // Capteur 1: Comptage + Début mesure vitesse
  if(digitalRead(voies[i].pinIR1)==LOW &&!voies[i].voitureEnCours){
    voies[i].compteurVoitures++;
    voies[i].t1 = millis();
    voies[i].voitureEnCours = true;
    Serial.print("Voie "); Serial.print(i); Serial.print(" +1 voiture. Total:");
    Serial.println(voies[i].compteurVoitures);
    delay(300); // Anti-rebond
  }
```

```
  if(digitalRead(voies[i].pinIR1)==HIGH){
    voies[i].voitureEnCours = false;
  }

  // Capteur 2: Fin mesure vitesse
  static unsigned long t2;
  if(digitalRead(voies[i].pinIR2)==LOW && voies[i].t1 > 0){
    t2 = millis();
    float tempsSec = (t2 - voies[i].t1) / 1000.0;
    float vitesseMS = DISTANCE_ENTRE_CAPTEURS / tempsSec;
    float vitesseKMH = vitesseMS * 3.6;

    Serial.print("Vitesse Voie "); Serial.print(i); Serial.print(": ");
    Serial.print(vitesseKMH); Serial.println(" km/h");

    // Détection Cortège
    if(vitesseKMH < VITESSE_CORTEGE_BASSE || vitesseKMH > VITESSE_CORTEGE_HAUTE){
      voies[i].priorite = true;
      Serial.print("!!! PRIORITE DETECTEE VOIE "); Serial.println(i);
    }
    voies[i].t1 = 0; // Reset
    delay(300);
  }
}
```

```
// --- GESTION CYCLE INTELLIGENT ---
void gererCycle(){
  // Vérifier priorité en premier
  for(int i=0; i<4; i++){
    if(voies[i].priorite && i!=voieActive){
      voies[i].priorite = false; // Consomme la priorité
      passerAuVert(i);
      return;
    }
  }

  // Calculer temps de vert adaptatif
  voies[voieActive].tempsVert = T_VERT_MIN + (voies[voieActive].compteurVoitures *
T_ADD_PAR_VOITURE);
  if(voies[voieActive].tempsVert > 120000) voies[voieActive].tempsVert = 120000; // Max 2min

  // Changer si temps écoulé
  if(millis() - chronoDepartVert > voies[voieActive].tempsVert){
    int prochaine = choisirProchaineVoie();
    passerAuVert(prochaine);
  }
}

void passerAuVert(int nouvelleVoie){
  // 1. Orange voie actuelle
  setFeu(voieActive, 0,1,0);
```

```
  delay(T_JAUNE_MIN);

  // 2. Rouge voie actuelle
  setFeu(voieActive, 0,0,1);
  delay(T_ROUGE_MIN); // Sécurité

  // 3. Vert nouvelle voie
  voieActive = nouvelleVoie;
  setFeu(voieActive, 1,0,0);
  voies[voieActive].compteurVoitures = 0; // Reset compteur après passage
  chronoDepartVert = millis();
}

int choisirProchaineVoie(){
  // Alterne entre Route 1 [0,1] et Route 2 [2,3]
  // Et prend la voie avec le plus de voitures
  int routeSuivante = (voieActive < 2)? 2 : 0;

  if(voies[routeSuivante].compteurVoitures >= voies[routeSuivante+1].compteurVoitures)
    return routeSuivante;
  else
    return routeSuivante + 1;
}

void gererPietons(){
```

```
  Serial.println("Demande Pieton");
  setFeu(voieActive, 0,1,0); // Orange
  delay(T_JAUNE_MIN);
  setFeu(voieActive, 0,0,1); // Rouge

  digitalWrite(ledPieton1, HIGH);
  digitalWrite(ledPieton2, HIGH);
  delay(15000); // 15s pour traverser

  digitalWrite(ledPieton1, LOW);
  digitalWrite(ledPieton2, LOW);
}

// --- UTILS ---
void setFeu(int voie, bool v, bool j, bool r){
  digitalWrite(voies[voie].pinV, v);
  digitalWrite(voies[voie].pinJ, j);
  digitalWrite(voies[voie].pinR, r);
}

void toutRouge(){
  for(int i=0; i<4; i++) setFeu(i,0,0,1);
}
```

Code pour les capteurs boucles magnétiques par Kambale kambere leonce

```
// Feu Tricolore Intelligent 2 Voies - Boucles Magnétiques
// Arduino Uno - Gestion: Comptage + Vitesse + Priorité + Piétons
// Temps min: V=60s, J=20s, R=60s

// --- CONSTANTES TEMPS ---
const unsigned long T_VERT_MIN = 60000; // 60s
const unsigned long T_JAUNE_MIN = 20000; // 20s
const unsigned long T_ROUGE_MIN = 60000; // 60s
const unsigned long T_ADD_PAR_VOITURE = 3000; // +3s par voiture
const unsigned long T_MAX_VERT = 120000; // Max 2min

// --- CONSTANTES DETECTION ---
const float DISTANCE_ENTRE_BOUCLES = 2.0; // 2 mètres entre boucle1 et boucle2
const float VITESSE_CORTEGE_BASSE = 10.0; // km/h deuil/mariage
const float VITESSE_CORTEGE_HAUTE = 60.0; // km/h autorités

// --- STRUCTURE VOIE ---
```

```
struct Voie {
  // LEDs Feu
  int pinV, pinJ, pinR;
  // Détecteurs de boucle: [0]=entrée, [1]=sortie
  int pinBoucle1, pinBoucle2;

  int compteurVoitures;
  bool priorite;
  unsigned long tempsVert;

  // Variables internes
  bool etatB1, etatB2, dernierEtatB1, dernierEtatB2;
  unsigned long t1; // temps détection boucle 1
  bool voitureComptee;
};

Voie voies[2]; // 0=VoieA, 1=VoieB
int voieActive = 0;
unsigned long chronoDepartVert;

// Pins Piétons
const int btnPieton = 7;
const int ledPieton = 6;

void setup() {
```

```
  Serial.begin(9600);

  // Init Voie A
  voies[0] = {8,9,10, 2,3, 0,false,T_VERT_MIN, 1,1,1,1, 0,false};
  // Init Voie B
  voies[1] = {11,12,13, 4,5, 0,false,T_VERT_MIN, 1,1,1,1, 0,false};

  // Init Pins
  for(int i=0; i<2; i++){
    pinMode(voies[i].pinV, OUTPUT);
    pinMode(voies[i].pinJ, OUTPUT);
    pinMode(voies[i].pinR, OUTPUT);
    pinMode(voies[i].pinBoucle1, INPUT_PULLUP); // Détecteur donne LOW si voiture
    pinMode(voies[i].pinBoucle2, INPUT_PULLUP);
  }

  pinMode(btnPieton, INPUT_PULLUP);
  pinMode(ledPieton, OUTPUT);

  toutRouge();
  delay(2000);
  passerAuVert(0);
}

void loop() {
```

```
  // 1. LIRE TOUTES LES BOUCLES
  for(int i=0; i<2; i++){
    lireBoucles(i);
  }

  // 2. GESTION PIETONS
  if(digitalRead(btnPieton)==LOW){
    gererPietons();
  }

  // 3. GESTION DU CYCLE
  gererCycle();
}

// --- LECTURE BOUCLES MAGNETIQUES ---
void lireBoucles(int i){
  voies[i].etatB1 = digitalRead(voies[i].pinBoucle1);
  voies[i].etatB2 = digitalRead(voies[i].pinBoucle2);

  // FRONT DESCENDANT Boucle1 = Voiture entre dans la voie
  if(voies[i].etatB1==LOW && voies[i].dernierEtatB1==HIGH &&!voies[i].voitureComptee){
    voies[i].compteurVoitures++;
    voies[i].t1 = millis();
    voies[i].voitureComptee = true;
    Serial.print("Voie "); Serial.print(i); Serial.print(" +1 voiture. Total:");
```

```
  Serial.println(voies[i].compteurVoitures);
  delay(100); // Anti-rebond
}

// FRONT MONTANT Boucle1 = Voiture partie
if(voies[i].etatB1==HIGH && voies[i].dernierEtatB1==LOW){
  voies[i].voitureComptee = false;
}

// FRONT DESCENDANT Boucle2 = Fin mesure vitesse
if(voies[i].etatB2==LOW && voies[i].dernierEtatB2==HIGH){
  if(voies[i].t1 > 0){
    unsigned long t2 = millis();
    float tempsSec = (t2 - voies[i].t1) / 1000.0;
    if(tempsSec > 0.1){ // Eviter division par 0
      float vitesseMS = DISTANCE_ENTRE_BOUCLES / tempsSec;
      float vitesseKMH = vitesseMS * 3.6;

      Serial.print("Vitesse Voie "); Serial.print(i); Serial.print(": ");
      Serial.print(vitesseKMH); Serial.println(" km/h");

      // Détection Cortège: Lent ou Rapide
      if(vitesseKMH < VITESSE_CORTEGE_BASSE || vitesseKMH > VITESSE_CORTEGE_HAUTE){
        voies[i].priorite = true;
        Serial.print("!!! PRIORITE DETECTEE VOIE "); Serial.println(i);
```

```
      }
     }
     voies[i].t1 = 0;
    }
    delay(100);
  }

  voies[i].dernierEtatB1 = voies[i].etatB1;
  voies[i].dernierEtatB2 = voies[i].etatB2;
}

// --- GESTION CYCLE INTELLIGENT ---
void gererCycle(){
  // 1. Vérifier Priorité en premier
  for(int i=0; i<2; i++){
    if(voies[i].priorite && i!=voieActive){
      voies[i].priorite = false;
      passerAuVert(i);
      return;
    }
  }

  // 2. Calculer temps de vert adaptatif
  voies[voieActive].tempsVert = T_VERT_MIN + (voies[voieActive].compteurVoitures *
T_ADD_PAR_VOITURE);
  if(voies[voieActive].tempsVert > T_MAX_VERT) voies[voieActive].tempsVert = T_MAX_VERT;
```

// 3. Changer si temps écoulé